2023.09.20

김주원

Attention is all you need

FOM

Focus On Data Mining

INDEX

Introduction

What's RNN

- RNN = Recurrent Neural Network (1986)
  - ✓ 시계열 데이터와 같이 순서가 있는 데이터 학습을 위한 인공신경망 구조

입력층의 벡터 x와 출력층의 벡터 y, 은닉층의 활성화 함수를 통해 결과를 내보내는 셀(cell)

✓ $h_t = \tanh(W_{hh}h_{t-1} + w_{xh}x_t + b)$

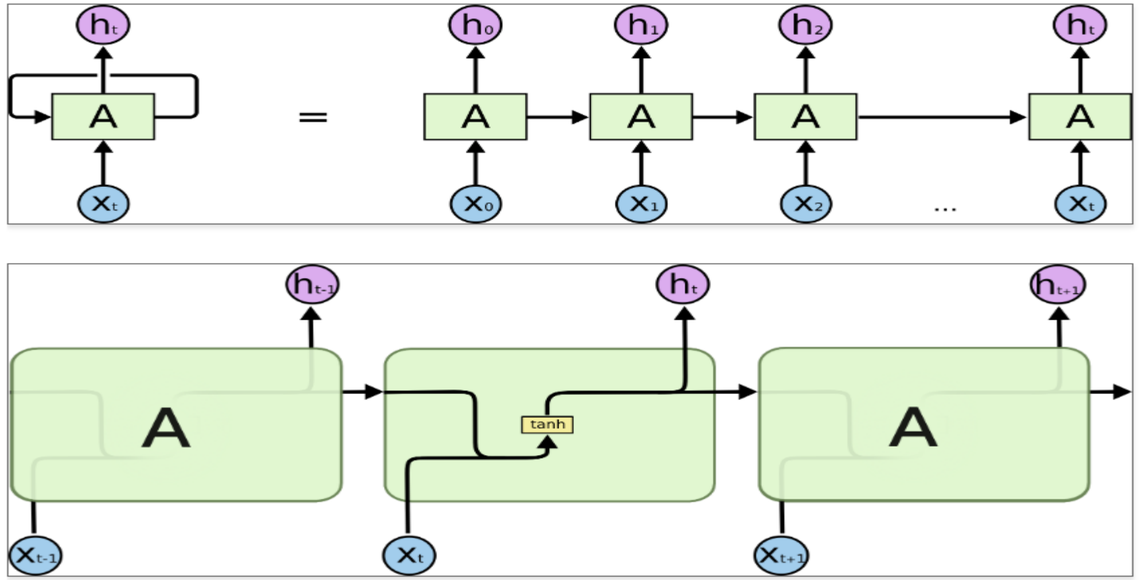
W_{hh} , W_{hh} = 가중치 행렬
h_{t-1} = 이전 시간 스텝의 은닉 상태

b : bias
h = 현재 시간의 은닉 상태

x = 입력

Introduction

What's RNN



Input x가 들어가면 여러 번의 순환을 거쳐 output인 y가 나오는 구조.

이전 단계에서 얻은 정보가 지속할 수 있도록 체인과 같은 구조로 tanh layer를 모듈로 사용.

Introduction

What's LSTM

- LSTM = Long Short-Term Memory (1997)

  ✓ RNN의 장기 의존성 문제 해결

  ✓ 메모리 셀과 게이트 메커니즘 도입

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

*ft,it,ot*는 각각 forget, input, output 게이트
σ는 시그모이드 함수, ⊙는 요소별 곱

02 | Relate Works

RNN & LSTM's problem

- Vanishing Gradient
  - ✓ 시퀀스가 길어지면 역전파 과정에서 Gradient가 소실될 수 있다
  - ✓ 시간을 거슬러 갈수록 점점 줄어드는 Gradient…

- Exploding Gradient
  - ✓ Gradient가 너무 커져서 불안정한 학습을 유발할 수 있다
  - ✓ If 가중치 행렬이 큰 값을 가진다면..?

02 | Relate Works

Long-term Dependencies

- RNN은 Long-term(장기) Dependencies(의존성)와 필연이다

Hani, who is foreign member and main vocalist of NewJeans, is pretty

✓ 기본RNN으로 위 문장을 처리해 봅시다

'Hani'에 대한 정보가 네트워크를 거치면서 점점 희석됨 ⇒ "is pretty" 부분에서 'Hani'에 대한 의존성을 정확히 학습하기 어려움

why) RNN은 기본적으로 직렬 구조를 지니므로 'Recurrent'란 이름답게 순차적으로 학습 (4p 그림을 참고해 주세요)

Long-term Dependencies를 해결하기 위한 새로운 매커니즘의 필요!

02 Relate Works

Appearance of Attention Mechanism

하나의 문맥 벡터 소스 문장의 모든 정보를 가지고 있어야 하므로 성능저하의 발생하는데..

매번 소스 문장에서의 출력 전부를 입력으로 받으면?

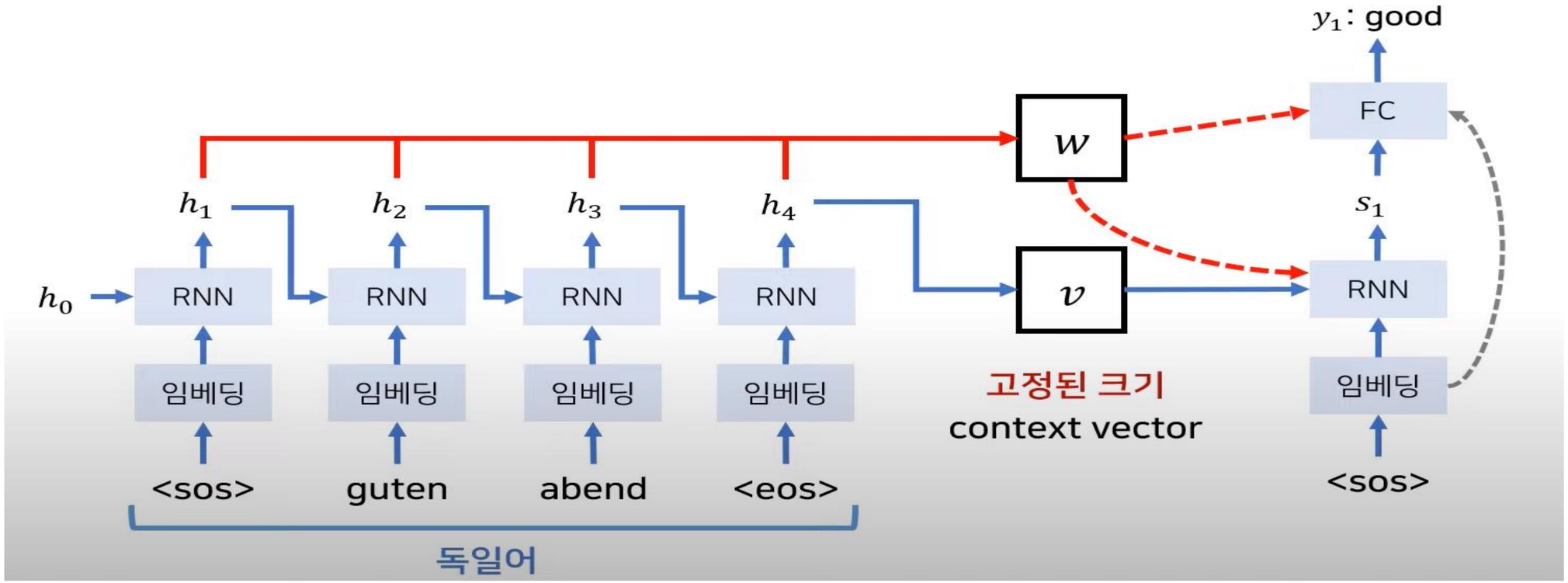
✓ 최신GPU(NVIDIA...)는 많은 메모리와 빠른 병렬처리 지원

Attention Mechanism의 등장!

02 Relate Works

Seq2Seq with Attention

- Seq2Seq모델에 Attention Mechanism 사용
  - ✓ 디코더는 인코더의 모든 outputs를 참고

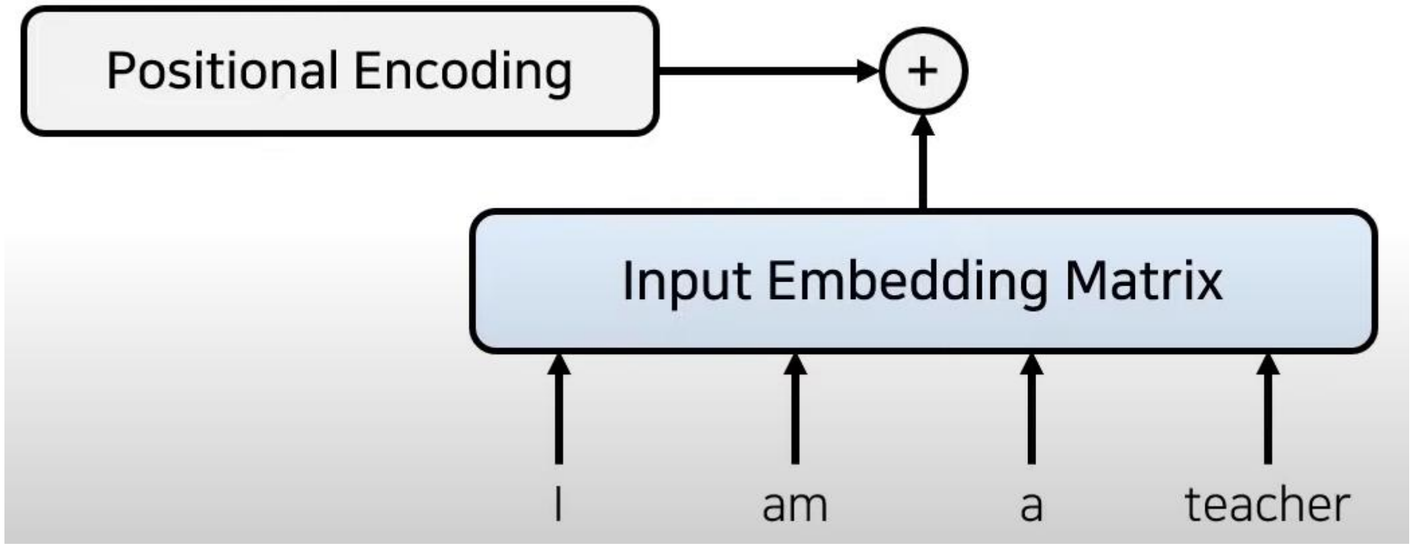


03 Proposed Method

Positional Encoding

- RNN을 사용하지 않기 위해 위치정보를 포함하는 임베딩 해줍니다

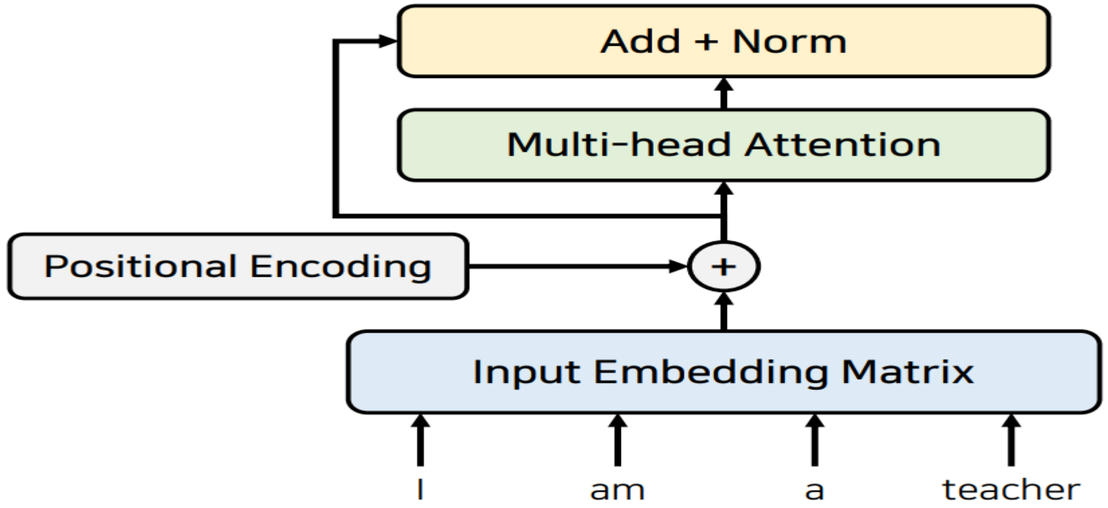
  ✓ Positional Encoding(위치 인코딩)에 위치정보를 넣어줘서 순서를 가짐을 알려줌



03 Proposed Method

Self attention & Residual Learning

- 위치정보를 포함한 입력값을 실제 Attention에 넣음(self attention)
  - ✓ 서로에게 Attention score를 줘서 문맥을 파악하게 됨
- 성능 향상을 위해 Residual Learning 실행
  - ✓ 특정 레이어를 건너뛰고 추가적으로 잔여 부분만 학습하게 되어 난이도와 속도를 잡는다

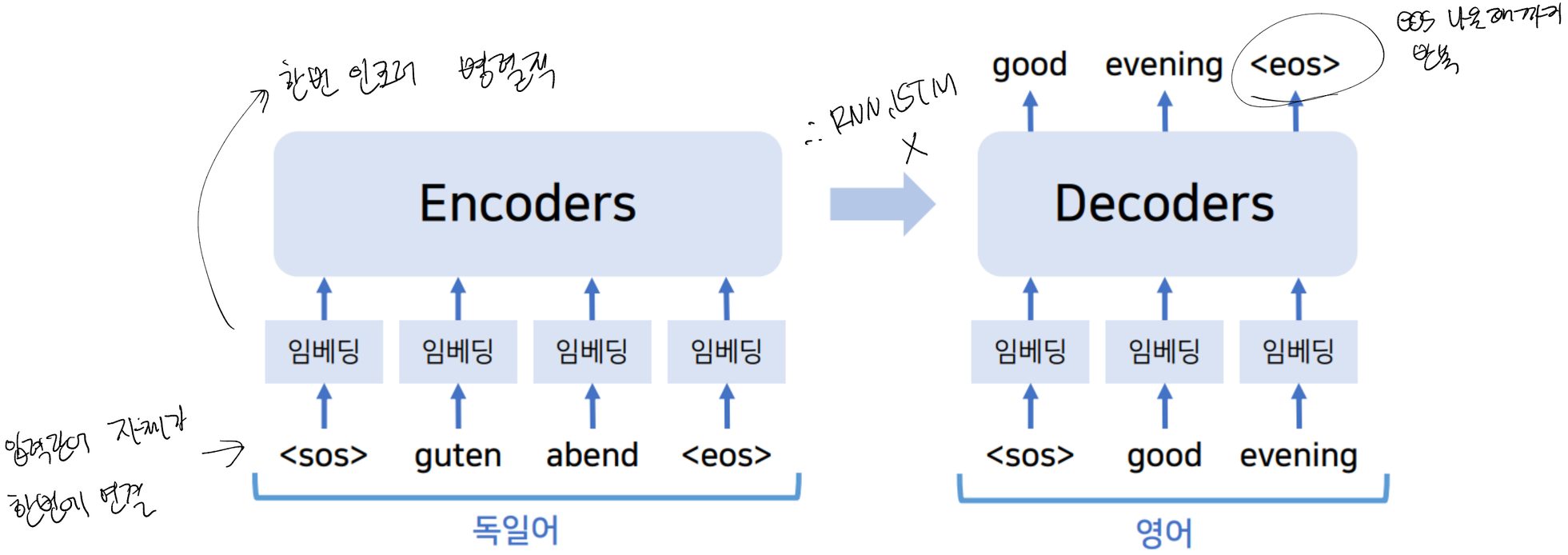


03 Proposed Method

Encoder & Decoder

- Transformer에서도 Encoder와 Decoder를 사용한다

✓ But RNN을 사용하지 않고 다수의 인코더와 디코더를 사용한다는 특징이 있음!



03 Proposed Method

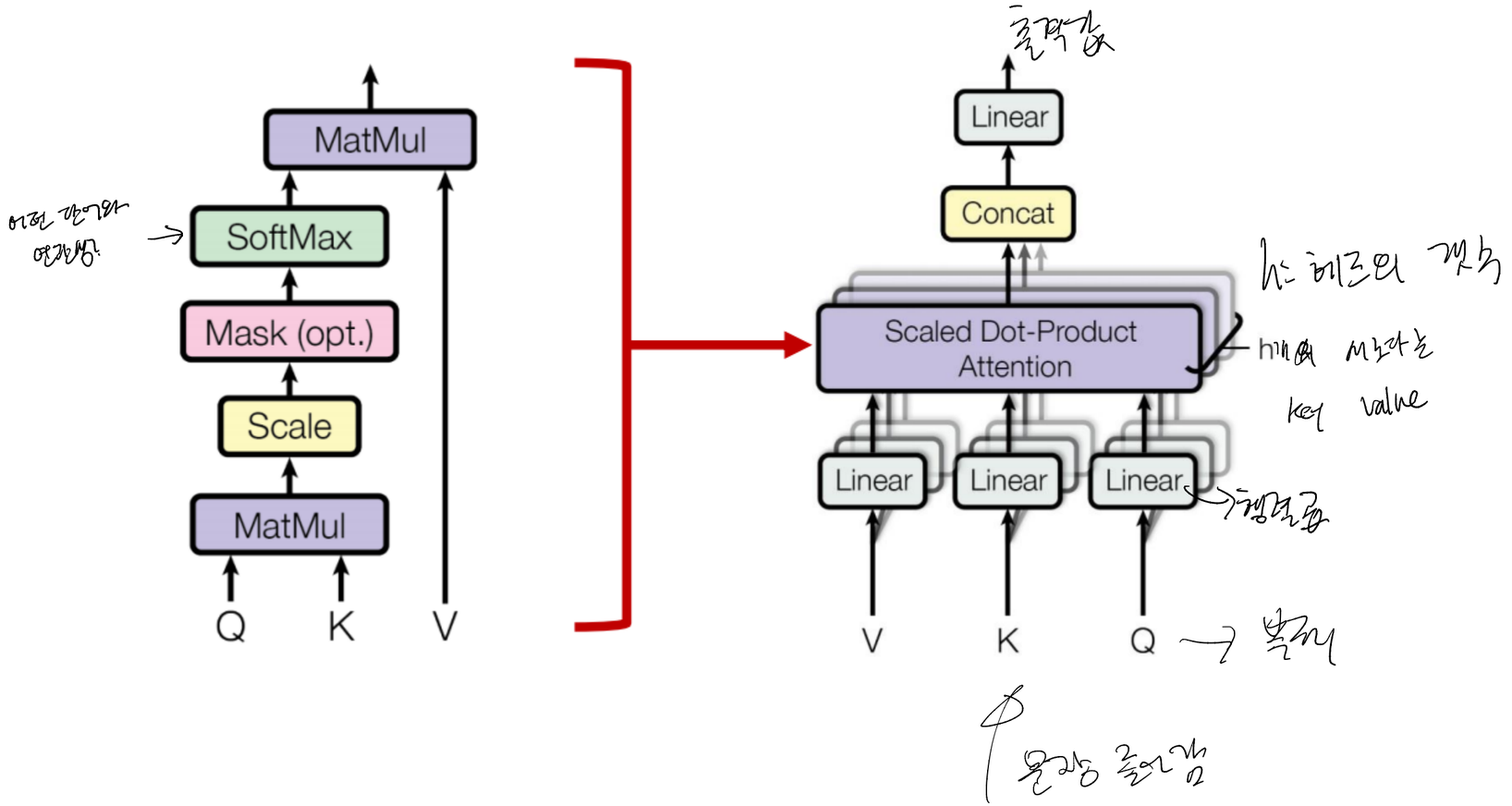
Query Key Value

- 인코더와 디코더는 Multi-Head Attention 레이어를 사용

✓ 쿼리 (Query)

✓ 키(Key)

✓ 값(Value)



03 | Proposed Method

What is Query

- Query

✓ 역할: 쿼리는 어텐션 메커니즘에서 중요한 정보를 선택하기 위한 "질문“ 의 역할

✓ 작동 원리: 쿼리는 키와 비교되어, 어떤 키-값 쌍이 현재 상황에 더 중요한지를 판단

✓ 예시: 문장에서 다음 단어를 예측하고자 할 때, 쿼리는 현재까지의 ‘문맥’을 나타낸다

03 | Proposed Method

What is Key

- Key

✓ 역할: 키는 데이터베이스에서 찾고자 하는 정보의 "주소" 역할

✓ 작동 원리: 각 키는 쿼리와 비교되어 그 호환성(또는 유사도)이 측정. 호환성은 각 값에 할당될 가중치를 결정하는 데 사용된다

✓ 예시: 문장 내의 다른 단어들이 키로 사용될 수 있으며, 이 키가 현재 문맥과 얼마나 관련이 있는지를 나타냄

03 | Proposed Method

What is Value

- Value

✓ 역할: 값은 키에 대응하는 실제 "정보"를 나타냄

✓ 작동 원리: 각 값은 쿼리와 호환성이 높은 키에 의해 선택되며, 최종적으로 이 값들의 가중 평균이 Attention의 출력으로 사용

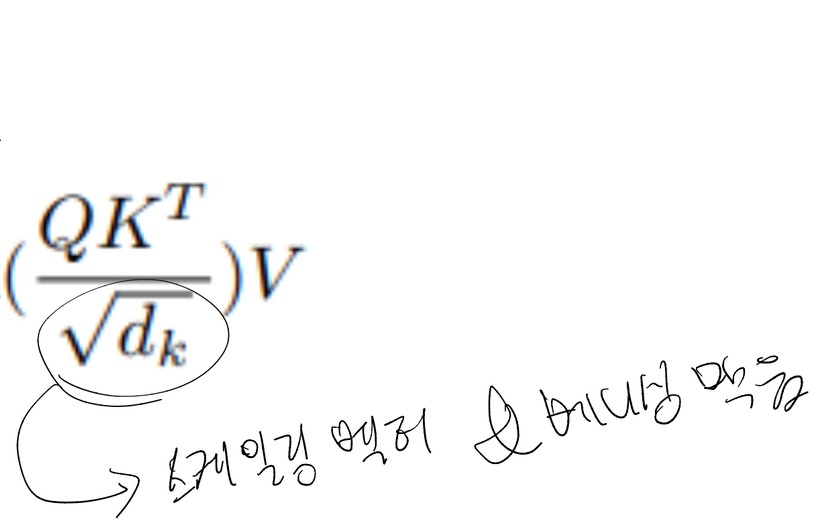
✓ 예시: 문장에서 다음 단어를 예측할 때, 각 단어의 값은 그 단어가 다음 단어가 될 확률에 기여하는 정보를 담고 있을 수 있다

03 | Proposed Method

수식

- Attention 수식

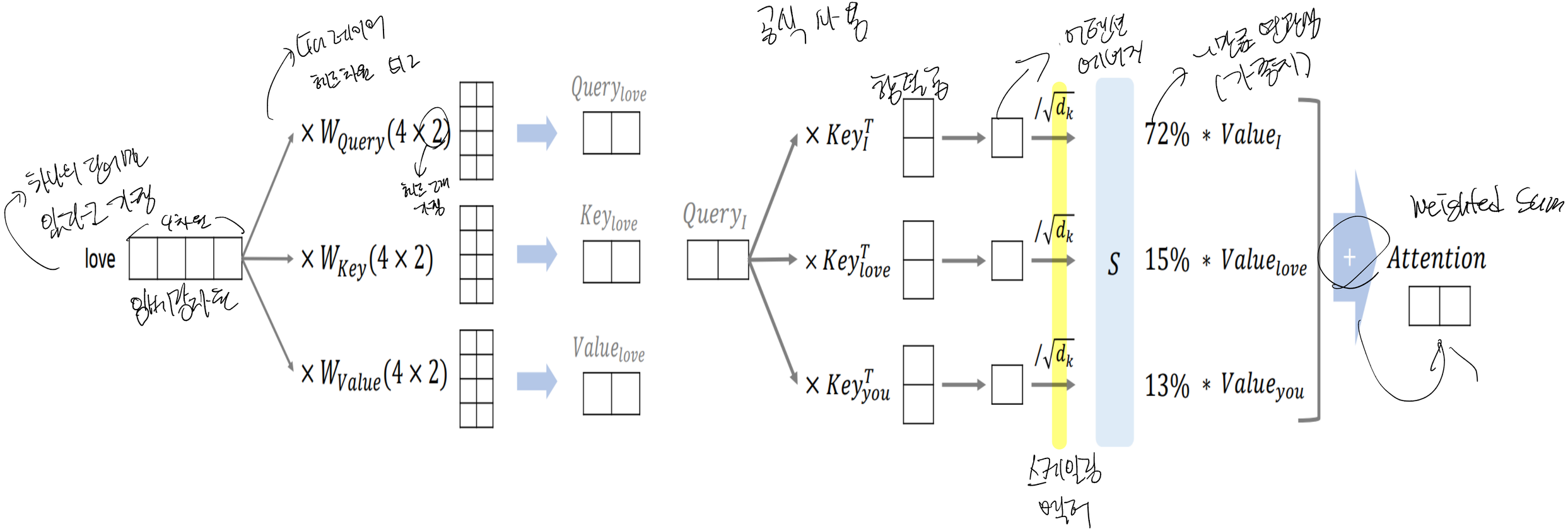
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



- ✓ d_k = Key와 Query의 차원
- ✓ Qk^T = 쿼리와 키의 내적을 계산하여 각각 큐와 키 사이의 유사도를 나타내는 행렬
- ✓ Softmax함수를 적용하여 유사도 값을 확률로 변환
- ✓ $\sqrt{d_k}$ = 스케일링 수행 -> 값의 범위를 제어하여 Softmax 함수가 더 잘 작동하게 함

03 Proposed Method

Attention



03 | Proposed Method

수식

- Multi-Head 수식

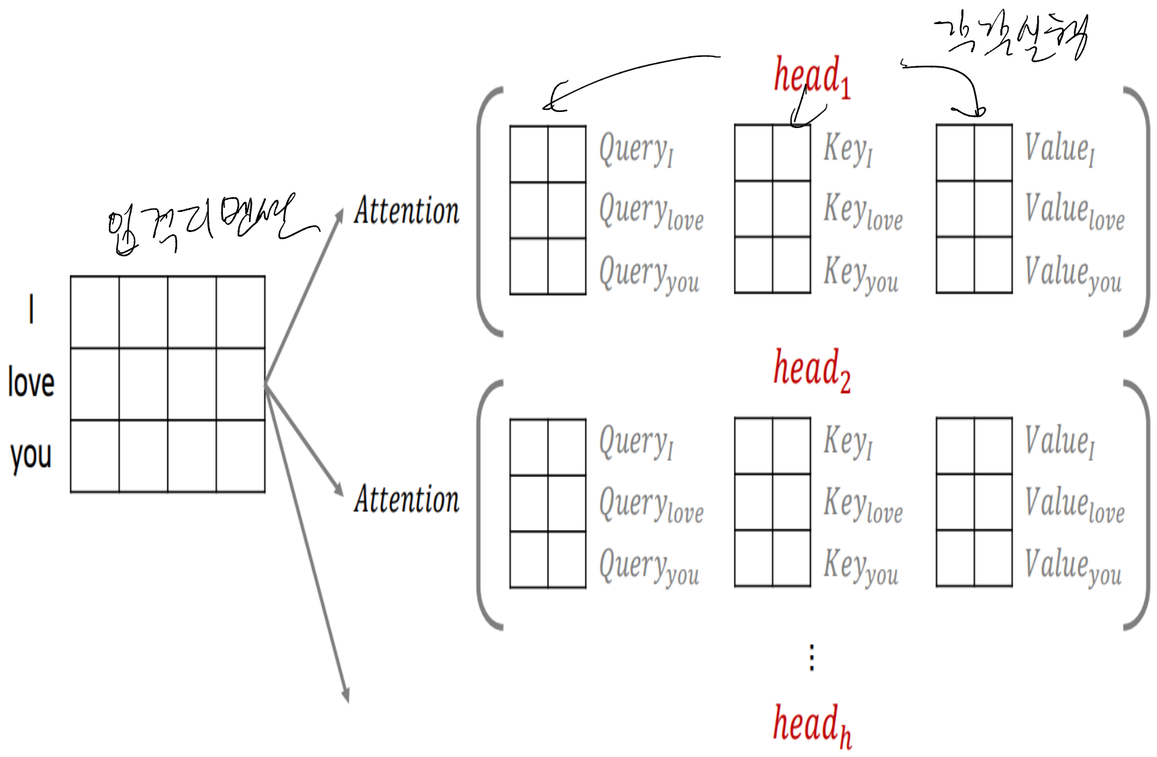
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$ → 다양한 패턴 학습

✓ QW, KW, VW는 학습 가능한 가중치 행렬

03 Proposed Method

Multi-Head



$$Concat(head_1, \ldots, head_h) = \underbrace{\begin{bmatrix} head_1 & head_2 & head_3 & \cdots & head_h \end{bmatrix}}_{d_{model} = d_v \times h}$$

$$MultiHead(Q,K,V) = \underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{d_{model} = d_v \times h} \; seq\_len \times \underbrace{\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}}_{d_{model}} \; d_{model}$$

03 Proposed Method

Transformer uses multi-head attention

- 인코더-디코더 어텐션

✓ 이 층에서의 쿼리(query)는 이전 디코더 층에서 나온다

✓ 메모리 키(keys)와 값(values)은 인코더의 출력에서 나옵니다. 이를 통해

✓ 디코더의 모든 위치가 입력 시퀀스의 모든 위치에 attend할 수 있음

seq2seq 모델에서 일반적으로 사용되는 인코더-디코더 Attention 메커니즘 사용!

03 Proposed Method

Transformer uses multi-head attention

- 인코더의 셀프 어텐션

✓ 인코더에는 self-attention 층이 포함.

✓ self-attention층에서는 키, 값, 그리고 쿼리가 모두 같은 위치

인코더의 이전 층의 출력에서 나오는데. 인코더의 각 위치는 이전 층의 모든 위치에 Attention할 수 있다!

03 Proposed Method

Transformer uses multi-head attention

- 디코더의 셀프 어텐션

✓ 마찬가지로 디코더의 셀프 어텐션 층은 디코더의 각 위치가 그 위치를 포함하여 그 이전의 모든 위치에 주목할 수 있게 한다.

✓ 여기서는 자동 회귀(auto-regressive) 속성을 유지하기 위해 디코더에서 좌측 정보 흐름을 차단해야..

✓ 이를 스케일드 닷-프로덕트 어텐션 내에서 구현하며, 이때 softmax의 입력에서 불법적인 연결에 해당하는 모든 값을 마스킹하여 음의 무한대로 설정

04 | Experiment

Machine Translation

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

✓ English-to-German Translation task에 대해 다른 모델들과 성능을 비교

✓ BLEU = 기계 번역 결과와 사람이 직접 번역한 결과가 얼마나 유사한지 비교

Transformer가 다른 모델들에 비해 높은 성능을 가지면서 Training cost가 가장 낮다!

04 | Experiment

Model Variation

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

✓ (B)key size d_k 를 너무 줄이면 quality가 떨어지고, C) 큰 모델이 더 성능이 좋다

✓ D) drop-out이 오버피팅을 피하는데 도움이 됨을 확인함

05 | Conclusion

Attention and NLP

- 2020년대 이후 가장 영향력 있는 자연어처리 관련 알고리즘
  - ✓ RNN, LSTM의 단점을 상쇄하며 이 셋을 전혀 사용하지 않음
  - ✓ 인코더, 디코더, 병렬구조
- 이후에 나오는 BERT & GPT의 시발점, 정말 안 쓰는 곳이 없다
  - ✓ 긴 문장과 대용량 데이터 처리에서 압도적인 성능!
- NLP뿐만 아니라 Vision 분야에도 영향
  - ✓ Vision 세션분들도 조금이나마 이번 발표가 도움이 되었다면 영광입니다

Q & A

감사합니다